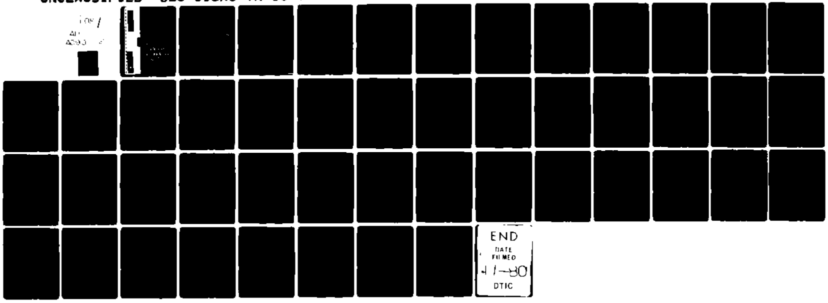
AD-A090 192 OHIO STATE UNIV COLUMBUS COMPUTER AND INFORMATION SC--ETC F/G 9/2
PARALLEL RECORD-SORTING METHODS FOR HARDWARE REALIZATION.(U)
JUL 80 D K HSIAO, M J MENON N00014-75-C-0573
UNCLASSIFIED OSU-CISRC-TR-80-7 NL



TECHNICAL REPORT SERIES

LEVEL III

(12)

AD A090192

DDC FILE COPY

COMPUTER & INFORMATION SCIENCE RESEARCH CENTER

80 8 27 058

THE OHIO STATE UNIVERSITY COLUMBUS, OHIO

(OSU-CISRC-TR-80-7)

(12)

Parallel Record-Sorting Methods for Hardware Realization

by

David K. Hsiao & M. Jaishankar Menon

Work Performed Under
Contract N00014-75-C-0573
Office of Naval Research

Computer and Information Science Research Center
The Ohio State University
Columbus, Ohio 43210
July 1980

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

14

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: OSU-CISRC-TR-80-7

2. GOVT ACCESSION NO.: AD-A090192

3. RECIPIENT'S CATALOG NUMBER:

4. TITLE (and Subtitle): 6 Parallel Record-Sorting Methods for Hardware Realization

5. TYPE OF REPORT & PERIOD COVERED: 9 Technical Report

6. PERFORMING ORG. REPORT NUMBER:

7. AUTHOR(s): 10 David K. Hsiao, M. Jaishankar Menon

8. CONTRACT OR GRANT NUMBER(s): 15 N00014-75-C-0573

9. PERFORMING ORGANIZATION NAME AND ADDRESS:
Office of Naval Research
Information Systems Program
Washington, D. C. 20360

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 784115-A1

11. CONTROLLING OFFICE NAME AND ADDRESS:

12. REPORT DATE: 11 Jul[redacted]80

13. NUMBER OF PAGES: 42

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): 12 47

15. SECURITY CLASS. (of this report):

15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Scientific Officer	DDC New York Area
ONR BRO	ONR 437
ACO	ONR, Boston
NRL 2627	ONR, Chicago
ONR 0121P	ONR, Pasadena

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Database computer, parallel sorting methods, record ordering, sort on value, post processing, post processing controller, odd-even sort, Stone sort, minimum-time sort.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

In this report, we demonstrate methods for implementing a fast hardware sorter in DBC. Given the trend towards cheaper processors and block-oriented access memories, we are especially interested in methods that utilize multiple processors with large-capacity block-oriented access memories in a parallel fashion.

Many parallel sorting methods are already available. Most of these methods use P processors to sort P records. Even though these methods are fast, they suffer from the fact that groups of records larger than P in number will have to

DD FORM 1 JAN 73 1473

407586

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered) be sorted in separate batches and then merged. Batching and merging defeats the intent of parallelism for high performance. Our methods, therefore, attempt to use P processors to sort more than P records without the need for batching and merging. Furthermore, our methods employ block-oriented access memories (such as magentic bubbles and charge-coupled devices).

Three sorting methods are described in this report. The first one, called the odd-even sort, uses P processors, each of which utilizes its own block-access memory to accomodate M records and has two processor-to-processor interconnections. This method sorts MP records in O(M log M + MP) time. There is no restriction on either P or M. More importantly, since each processor needs to be connected to only two others, we can increase the number of processors in the sorter without having to increase the number of connections per processor.

The second method, called the modified Stone sort, uses P processors with block-access memory to sort MP records in $O(M \log M + M \log^2 P)$ time. However, P must be a power of two. This method also requires that each processor be connected to a maximum of two others, one connection being a one-way link and the other connection being a two-way link.

Finally, we discuss a specialized minimum-time sort method. This method also uses P processors to sort MP records and completes the sorting in O(M log M) time, but it is used only for specific values of P. In other words, the method is individually tailored for a given number of processors. Once the design is completed, the sorter will not work for different values of P. Although the method so tailored is optimal for given values of P, the architecture of these types of hardware sorters cannot be expanded, since the number of processors is fixed for the original design and optimization.

In all the above methods, the number of records that can be sorted in a batch is restricted only by the memory size of each processor and not by the number of processors.

PREFACE

This work was supported by contract N00014-75-C-0573 from the Office of Naval Research to Dr. David K. Hsiao, Professor of Computer and Information Science, and conducted at the Computer and Information Science Research Center of The Ohio State University. The Computer and Information Science Research Center of The Ohio State University is an interdisciplinary research organization which consists of the staff, graduate students and faculty of many University departments and laboratories. This report is based on research accomplished in cooperation with the Department of Computer and Information Science. The research contract was administered and monitored by The Ohio State University Research Foundation.

TABLE OF CONTENTS

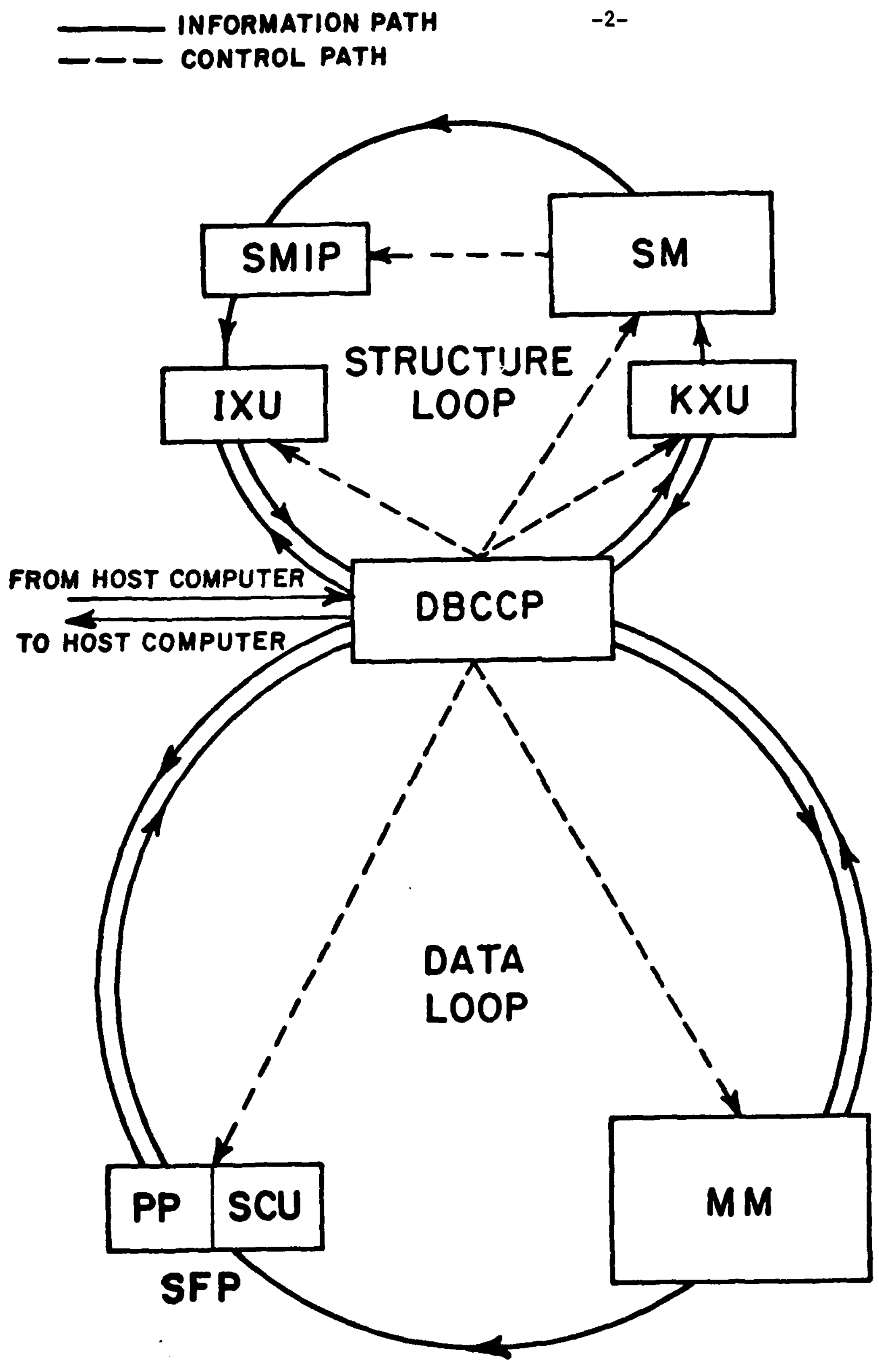
1. BACKGROUND

In the previous technical reports and papers [1-10], we presented various aspects of the design of a database computer known as DBC. In this report, we intend to demonstrate three methods for implementing a fast hardware sorter in DBC. The motivation for using a fast hardware sorter in DBC and for introducing an early method is described in [8]. We shall not repeat the motivation here. The comparison of these methods with the early one, however, constitutes a part of this report.

The architecture of DBC is shown in Figure 1. We are especially interested in the portion of the security filter processor (SFP) known as the post processor (PP). The post processor of DBC has the capability for performing the relational join operation and for computing set functions (maximum, minimum, average, counting and summation) [9]. Additionally, it is capable of sorting records. Sorting methods which use a single processor to sort N records have long been available. Some of these methods employ random-access memories and others employ sequential-access memories. Given the present trend towards charge-coupled devices and magnetic bubble memories which are essentially block-oriented sequential-access memories (or block-access memories), we are more interested in sorting methods which use block-access memory. One such method which commonly uses a single processor and a linear amount of sequential memory, i.e., O(N) memory, to sort N elements in O(N log N) time, is an adaptation of the merge-sort method of Knuth [11, pp. 159-168].

It is becoming increasingly obvious that the cost of processors will continue to fall in the future. Given this trend, it is natural that we should try to exploit fairly cheap processors by designing a sorting method that uses many parallel processors. Such sorting methods are already available [8, 12, 13, 14, 15, 16, 17]. All these methods, with the exception of [8, 16, 17], use P processors to sort P records. Therefore, even though these methods are fast, they suffer from the fact that groups of records larger than P in number will have to be sorted in separate batches of P records and then be merged. Thus, these methods are _processor limited_, i.e., the number of records that can be sorted is limited by the number of processors.

The methods of [8, 16, 17] all use P processors, each with O(M)



DBCCP:	Data Base Command Control Processor
KXU:	Keyword Transform Unit
SM:	Structure Memory
SMIP:	Structure Memory Information Processor
IXU:	Index Translation Unit
MM:	Mass Memory
SFP:	Security Filter Processor
SCU:	Security and Clustering Unit
PP:	Post Processor

Figure 1. The Architecture of DBC

sequential memory, to sort MP records. The method of [16] uses P processors, each with sequential memory of size M, to sort MP records in $O(M^2P)$ time. Also, the memory attached to a processor is a single loop of magnetic bubble or charge-coupled device memory. Thus, M is restricted by the size of a single loop which would probably be small. Hence, a large number of records cannot be sorted in a batch.

The methods of [17] use P processors, each with sequential memory of size 5M, to sort MP records in O(M log M) time. What makes these methods unattractive is the need for a large amount of memory. Thus, even though each processor memory holds only M records, it needs an additional 4M memory as workspace. The amount of workspace needed may be reduced to 3M with clever coding, though the authors do not point this out.

The method of [8] uses P processors, each with sequential memory of size 2M, to sort MP records in O(M log M) time. However, this method has the following limitations:

(1) As many as log P (all logarithms in this report are of base two) direct interconnections are needed between processors. That is, each processor needs to be connected directly to log P others.

(2) Once the number of processors, i.e., P is chosen, the hardware structure is not extensible. Let us elaborate this limitation. Consider that a hardware sorter is built using 16 parallel processors, where each processor is connected directly to four, i.e., log 16, others. Therefore, each processor memory is designed to have four ports in order that it may be connected to four other processors. Now, consider that, at a later time, we wish to have 32 parallel processors in the hardware sorter. Now, each processor must be connected to five other processors. However, if there are not enough ports in the processor memories, such interconnections are not possible.

(3) The number of processors that need to be used in this method must be a power of two.

The methods which we will describe in this report all use P processors, each with block-access memory of size 2M, to sort MP records in O(M log M) time. Thus, these methods are not processor limited. Furthermore, some or all of the three limitations listed above are eliminated from these methods.

2. OBJECTIVES

Our objectives are therefore to design a sorting method for hardware realization using P parallel processors with the following properties:

(1) P, i.e., the number of processors, need not be a power of two.

(2) It can sort groups of records larger than P in number without the need to sort them in separate batches for subsequent merges.

(3) It uses only block-access memory and does not need random-access memory.

(4) It uses fewer than log P direct connections per processor.

(5) It has an extensible hardware structure.

We propose three hardware schemes in the subsequent sections of this report. The first method meets all the five objectives listed above, but is slower than the method suggested in [8]. The second method meets all but the first objective listed above and is as fast as the method suggested in [8]. The third method meets all but the last objective and is faster than the other two methods.

3. TERMINOLOGY AND PROBLEM STATEMENT

To facilitate our discussion, we introduce some terminology. All the sorting methods suggested in this report use P parallel processors, where each processor has attached to it enough memory to accomodate M records. An additional workspace of size M is needed per processor, but we will not repeat this point anymore since it is not pertinent to the presentation of the algorithms. Altogether there are MP records. The processors are numbered 0 through (P-1) and the M records in Processor i's memory for $0 \leq i \leq (P-1)$ are named R[i,1], R[i,2],..., R[i,M]. Figure 2 depicts the record layout scheme. We make the simplifying assumption that records are of fixed-length. Furthermore, let us suppose that one block of memory can hold exactly one record. Thus, each processor's memory has M blocks in it and there are MP blocks altogether. Each record is composed of a number of attribute-value pairs. Whenever a record set is to be sorted, it is meant that the records of the set are ordered on the basis of increasing (or decreasing) values of a certain attribute of the set. Thus, to sort a record set, an attribute of the set must be designated first as the <u>sort attribute</u>. The <u>sort values</u> are those attribute values of the sort attribute. The number of sort values (not all of them being distinct) is equal to the cardinality

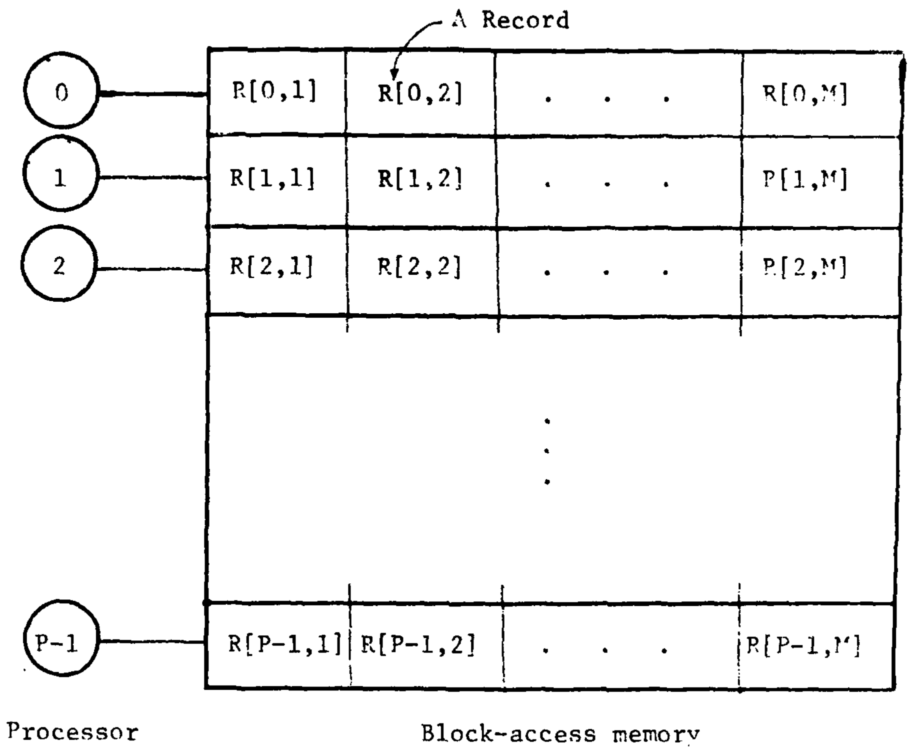


FIGURE 2. Record Layout Scheme When There Are P Processors and M Records Per Processor

of the record set. For example, in Figure 3, we illustrate a set of three records each of which consists of attribute-values on rank, age, salary, etc. We may sort these records on a number of attributes. If the chosen sort attribute is Rank, then we may represent the records with their rank values, namely, 5 for Record 1, 3 for Record 2 and 9 for Record 3. We do not have to show other attribute values of the records in representation, because these other attribute values are not in consideration. (See Figure 3 again). Upon sorting, Record 1 should precede Record 3 and follow Record 2, since this is the sequence of the sort values. On the other hand, if we choose either Age or Salary as the sort attribute, the sequence of sorted records will be different. More specifically, Record 1 will be last and Record 3 will be first in the sequence. This sequence of records is dictated by either ages or salaries. (see Figure 3 once more). This example shows that we represent records with their sort values. For simplicity, we eliminate the other attribute values. In the following sections, we shall use positive integers to represent the sort values of the records. A large rectangle represents the memory attached to a processor and a small rectangle a memory block. The integer within the block represents a record. Furthermore, we shall use circles to denote the processors. In Figure 4, there are two processors, namely, Processor 0 and Processor 1. Their corresponding block-access memories are represented by rectangular boxes. For example, initially there are five records in Processor 0's memory and another five records in Processor 1's memory (one record to a block). The sort values of the records in Processor 0's memory are 3, 9, 8, 2 and 17 whereas the sort values of the records in Processor 1's memory are 6, 2, 1, 3 and 5.

4. PRINCIPLE OF OPERATION

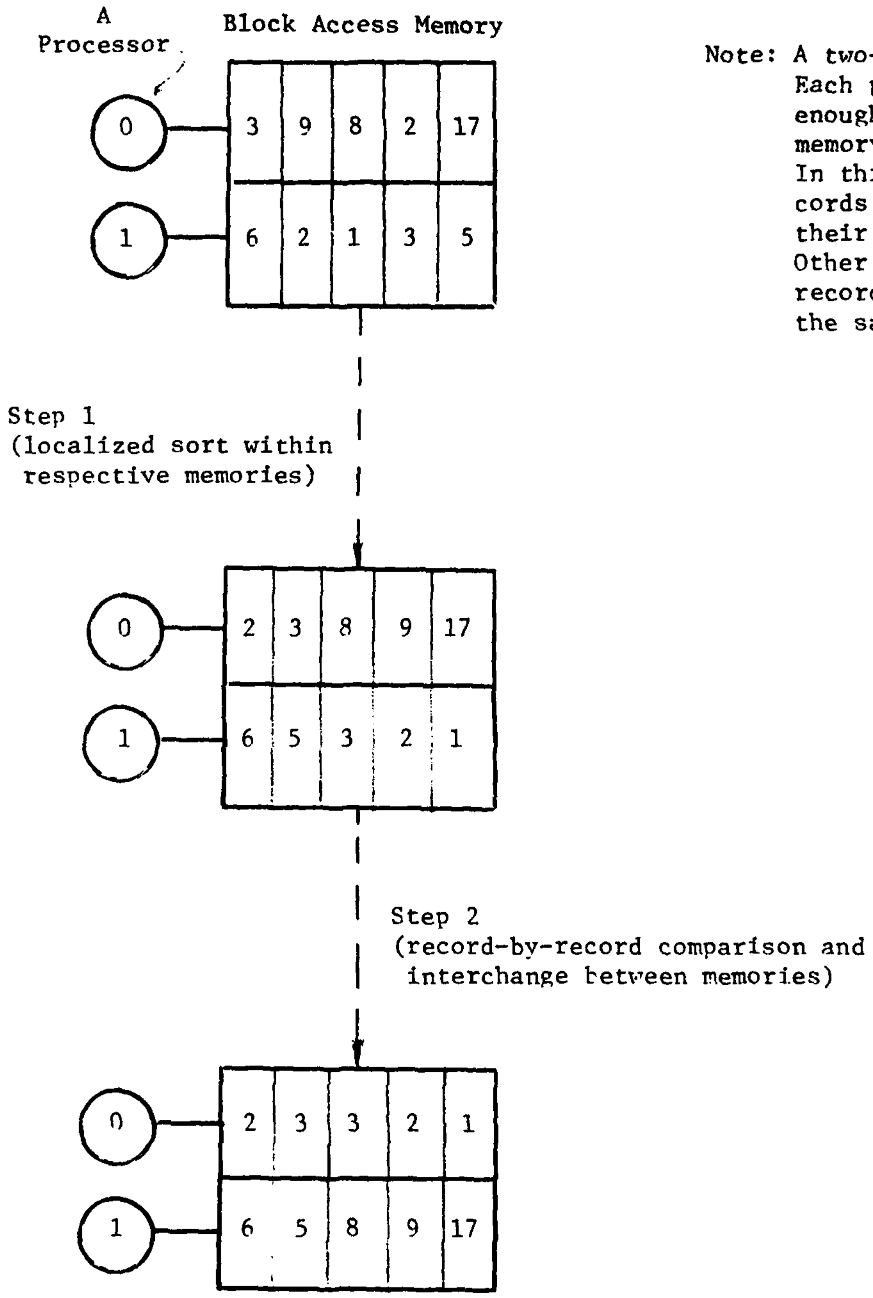
We would like to sort the 2M records in two processor memories on a certain sort attribute in such a way that all M records that appear in Processor 0's memory have their sort values lower than any sort value of the M records that appear in Processor 1's memory. We shall now detail the method.

First, find the largest record (i.e., the record with the largest sort value for the sort attribute of those records in the same memory) in Processor 0's memory and also find the smallest record (i.e., the record with smallest sort value for the sort attribute) in Processor 1's memory. If

	Actual Records with Record #'s attached at the upper left corners	Same Records Represented by Different Sort Values		
		Sort Values on Rank	Sort Values on Age	Sort Values on Salary
1	Rank 5 Age 60 Salary 30,000 Job Professor ⋮	5	60	30,000
2	Rank 3 Age 35 Salary 21,000 Job Analyst ⋮	3	35	21,000
3	Rank 9 Age 28 Salary 18,500 Job Clerk ⋮	9	28	18,500

Prior to any sorting

FIGURE 3. Representing Records by Sort Values.



A Processor
Block Access Memory
0
1
3 9 8 2 17
6 2 1 3 5
Step 1
(localized sort within respective memories)
2 3 8 9 17
6 5 3 2 1
Step 2
(record-by-record comparison and interchange between memories)
2 3 3 2 1
6 5 8 9 17

Note: A two-processor system. Each processor has enough block-access memory to hold M records. In this case, M = 5. Records are represented by their sort values alone. Other information of the records are removed for the sake of simplicity.

FIGURE 4. A Two-Step Exchange Process

the largest record in Processor 0's memory is less than or equal to the smallest record in Processor 1's memory, there is no need to exchange. Otherwise, we exchange the record in one memory with the record in the other memory. Next, find the second largest record in Processor 0's memory and the second smallest record in Processor 1's memory and compare the sort attributes of the two records. Once again, if the second largest record in Processor 0's memory is less than or equal to the second smallest record in Processor 1's memory, there is no need to exchange. Otherwise, the records are exchanged. The process continues until no more exchanging is needed. Thus, all M records in Processor 0's memory are smaller in sort values than the sort values of the M records in Processor 1's memory.

The entire process of exchanging records may be done as follows. First, Processor 0 sorts the records in its own memory in ascending order of the sort values. At the same time, Processor 1 sorts the records in its own memory in descending order of the sort values. Now, Processors 0 and 1 compare and exchange, if necessary, corresponding records, i.e., the first record of Processor 0 (the record with smallest sort value in Processor 0's memory) with the first record of Processor 1 (the record with largest sort value in Processor 1's memory), the second record of Processor 0 (the record with the second smallest sort value in Processor 0's memory) with the second record of Processor 1 (the record with the second largest sort value in Processor 1's memory), and so on. At the end of the exchange process, the sort values of all the M records in Processor 0's memory are smaller than any sort value of the M records in Processor 1's memory. The aforementioned steps are illustrated in Figure 4. The idea is based on the discovery by Alekseyev [18] that in order to select the largest t elements out of 2t elements $< x_1, x_2, \dots, x_{2t} >$, we may first sort $< x_1, x_2, \dots, x_t >$ and then sort $< x_{t+1}, x_{t+2}, \dots, x_{2t} >$ and then compare and interchange x_1 with x_{2t}, x_2 with $x_{2t-1}, \dots, x_t$ with x_{t+1}. After records are exchanged between the memories in the manner described above, let each processor do a localized sort of records in its own memory on the basis of the sort values. Now all 2M records will have been sorted on the basis of the sort values.

We now summarize the above discussion by adopting the following notation.

EXCHANGE [i,j] - This means the exchange of the records in Processors i and j in such a way that the smallest M records are in Processor i and the largest M records are in Processor j. This, as we know, is done in the following manner. First, Processor i sorts its own records in ascending order. At the same time, Processor j sorts its own records in descending order. Now Processors i and j compare corresponding records and the smaller records are placed in Processor i and the larger records in Processor j.

We note, that, at the end of EXCHANGE [i,j], we have a bitonic sequence in Processor i's memory and a bitonic sequence in Processor j's memory.

A sequence $S = (S_1, S_2, \ldots S_m)$ is <u>bitonic</u> if there is an <u>index</u> k, where $1 < k < m$ such that either

(1) $S_1 \le S_2 \le \ldots \le S_k \ge S_{k+1} \ldots \ge S_m$ or

(2) $S_1 \ge S_2 \ge \ldots \ge S_k \le S_{k+1} \ldots \le S_m$

The process EXCHANGE [0,1] has been shown, as a two-step process, in Figure 4. In Figure 5, the effect of EXCHANGE [0,1] has resulted in two bitonic sequences. The index record in Processor 0's memory is the record having the sort value 3. The index record in Processor 1's memory is the record having the sort value 5.

Let us now generalize the two processor-memory pairs to a sorter with P processor-memory pairs. The sorter uses Processors 0, 1, 2, ..., (P-1) to sort MP records (where M is the number of records that can be stored in a processor's memory) in such a way that the smallest M records are in Processor 0, the second smallest M records are in Processor 1, and so on. This is accomplished by a number of EXCHANGE [i, j]'s. Each processor is then to sort the records in its own memory. Thus the entire MP records are in sort order.

The sorting methods suggested in the following sections are variations of the above method. They all use P parallel processors, where each processor has enough memory to accomodate M records. None of these methods are processor limited.

5. METHOD I - THE ODD-EVEN SORT

The method is a generalization of the odd-even transposition sort

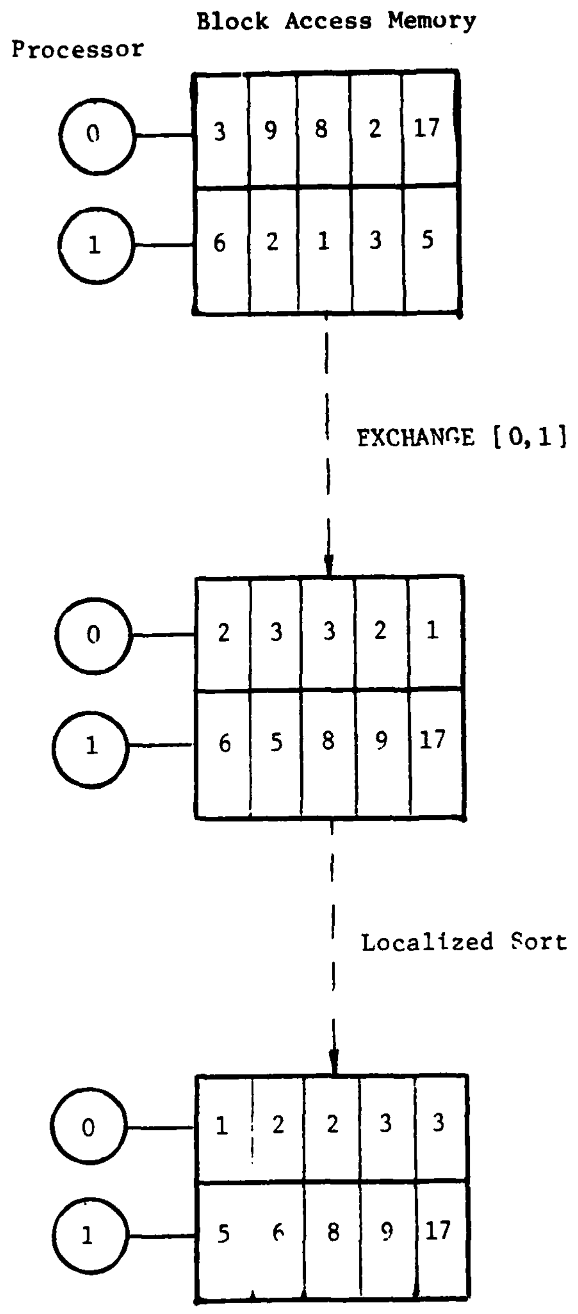


Figure 5. Sorting 10 Records Using Two Processor-Memory Pairs.

[11, pg. 241] and is illustrated in Figure 6, which needs more explanation. Consider, for illustration, the case where P, the number of the processors, is 4. We may imagine that the records enter at the left side of the diagram. The records in the memory attached to Processor 0 enter at the line marked 0, the records in the memory attached to Processor 1 enter at the line marked 1, and so on. The records then go through four steps as shown in the figure, and emerge at the right end. The vertical lines in each step, represent an EXCHANGE operation. A vertical line from i to j represents EXCHANGE [i, j]. The four steps are as follows. The first step consists of parallel operations of EXCHANGE [0, 1] and EXCHANGE [2, 3]. The second step consists of an EXCHANGE [1, 2]. The third step is similar to the first step, and the fourth step is similar to the second step. Finally, each processor performs a localized sort (this is not shown in the figure). Looking at Figure 6, we see that the number of steps is exactly equal to the number of processors used. That is, the method uses P steps followed by a final stage where each processor performs a localized sort.

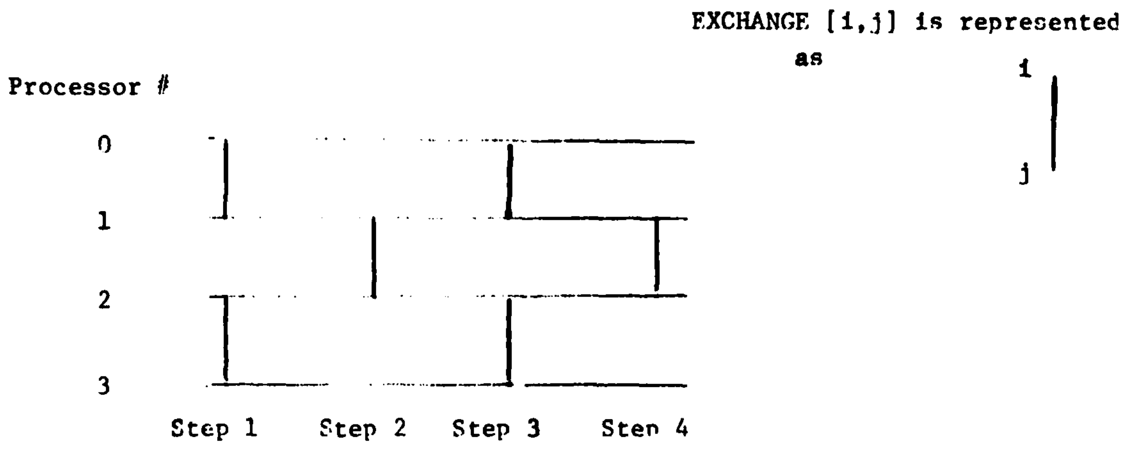
5.1 An Illustration

The manner in which the P parallel processors sort MP records is illustrated by means of an example developed in Figure 7. In this case, P = 4 and M = 5. The initial configuration of records (represented by sort values alone) is shown in Figure 7a. Four steps plus a localized sort are involved in the process.

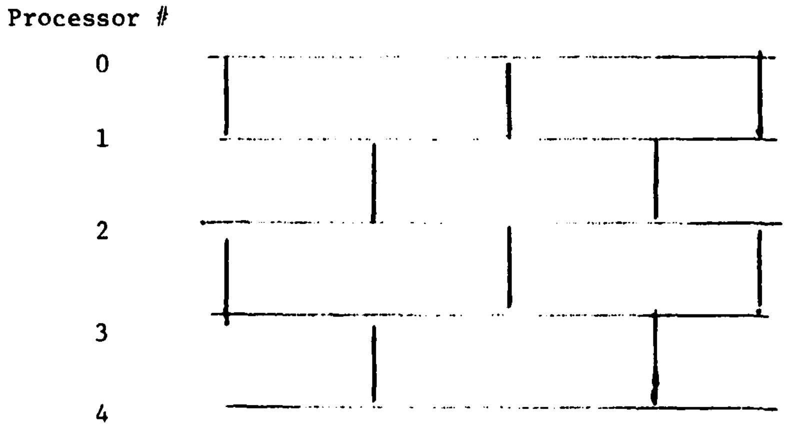
Step 1: EXCHANGE [0,1], EXCHANGE [2,3]
Step 2: EXCHANGE [1,2] (See Figure 7b.)
Step 3: EXCHANGE [0,1], EXCHANGE [2,3] (See Figure 7c.)
Step 4: EXCHANGE [1,2] (See Figure 7d.)
Step 5: Each processor sorts its own records in non-decreasing order. (See Figure 7e.)

5.2 The Algorithm

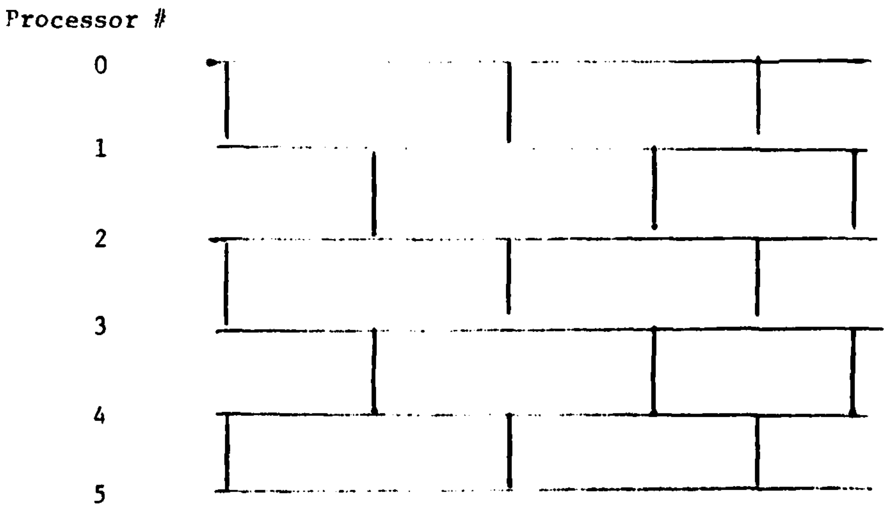
The parallel odd-even sorting algorithm is coordinated by a post processing controller (PPC). It broadcasts commands to all the processors. The following variables are used in the procedures that con-



case where P = 4

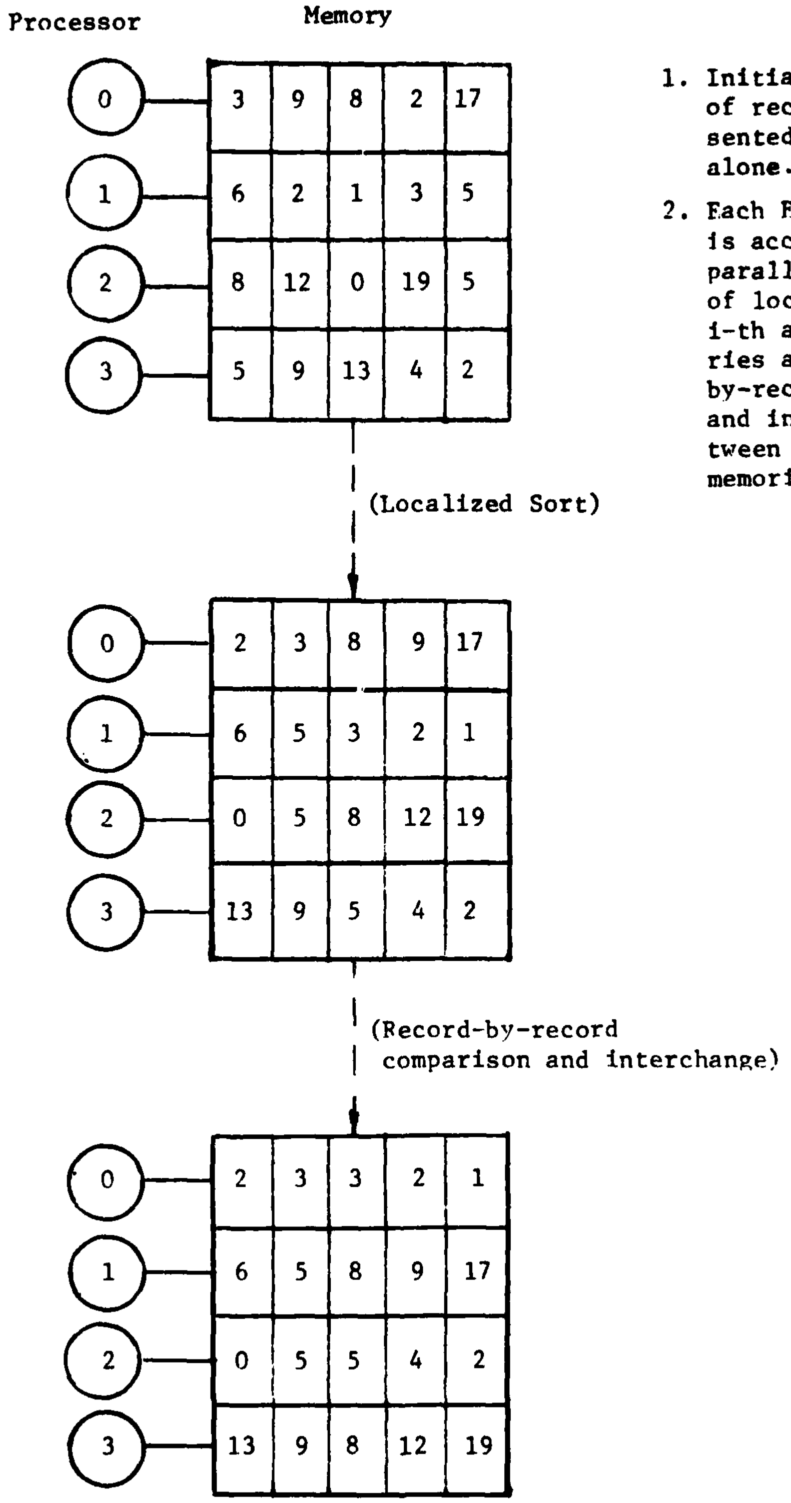


case where P = 5



case where P = 6

FIGURE 6. The ODD-EVEN Sort



1. Initial configuration of records are represented by sort values alone.
2. Each EXCHANGE [i,j] is accomplished by parallel operations of localized sort in i-th and j-th memories and by record-by-record comparison and interchange between i-th and j-th memories.

FIGURE 7a. Step 1: EXCHANGE [0,1] and EXCHANGE [2,3]

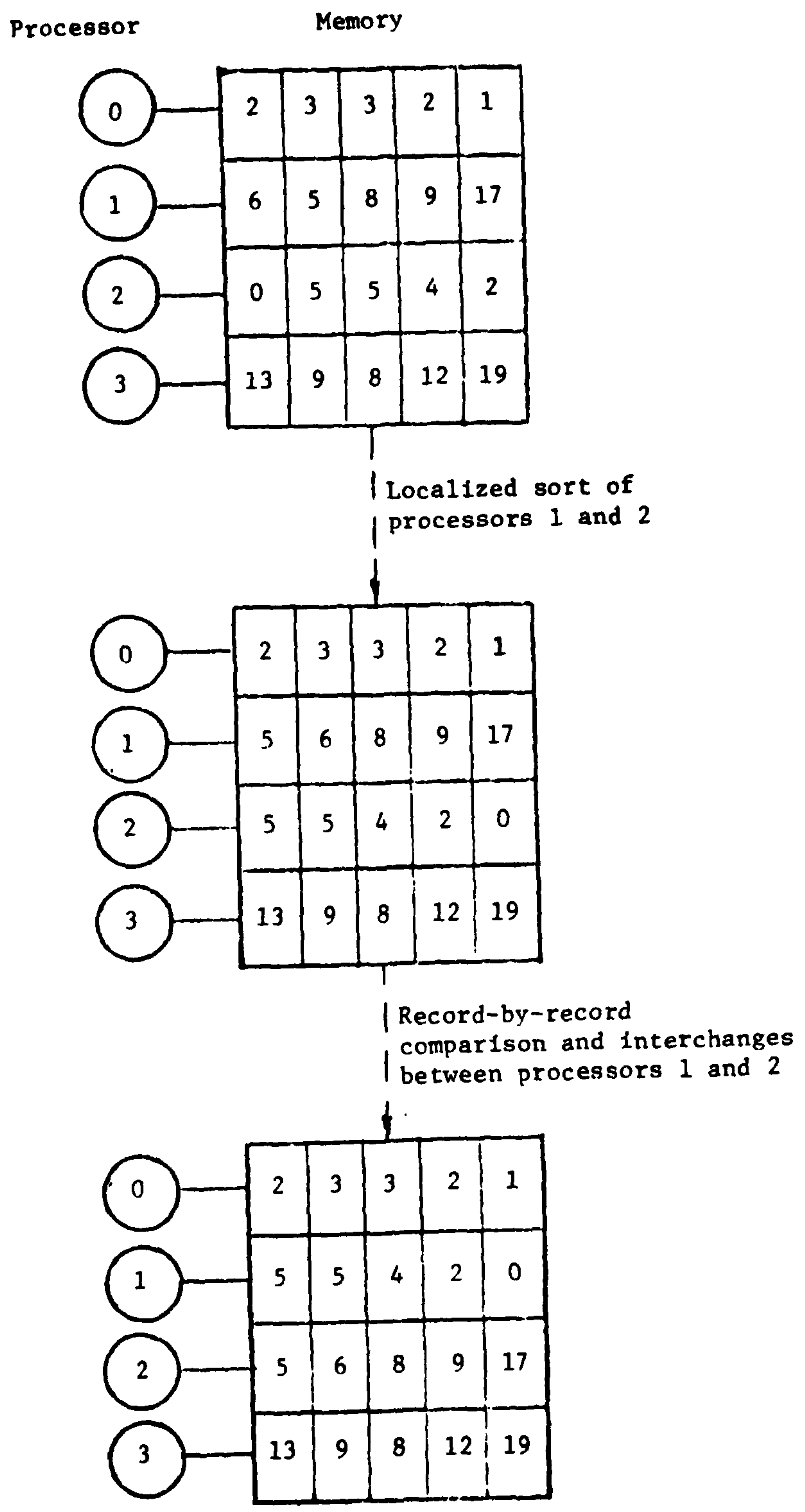


FIGURE 7b. Step 2: EXCHANGE [1,2]

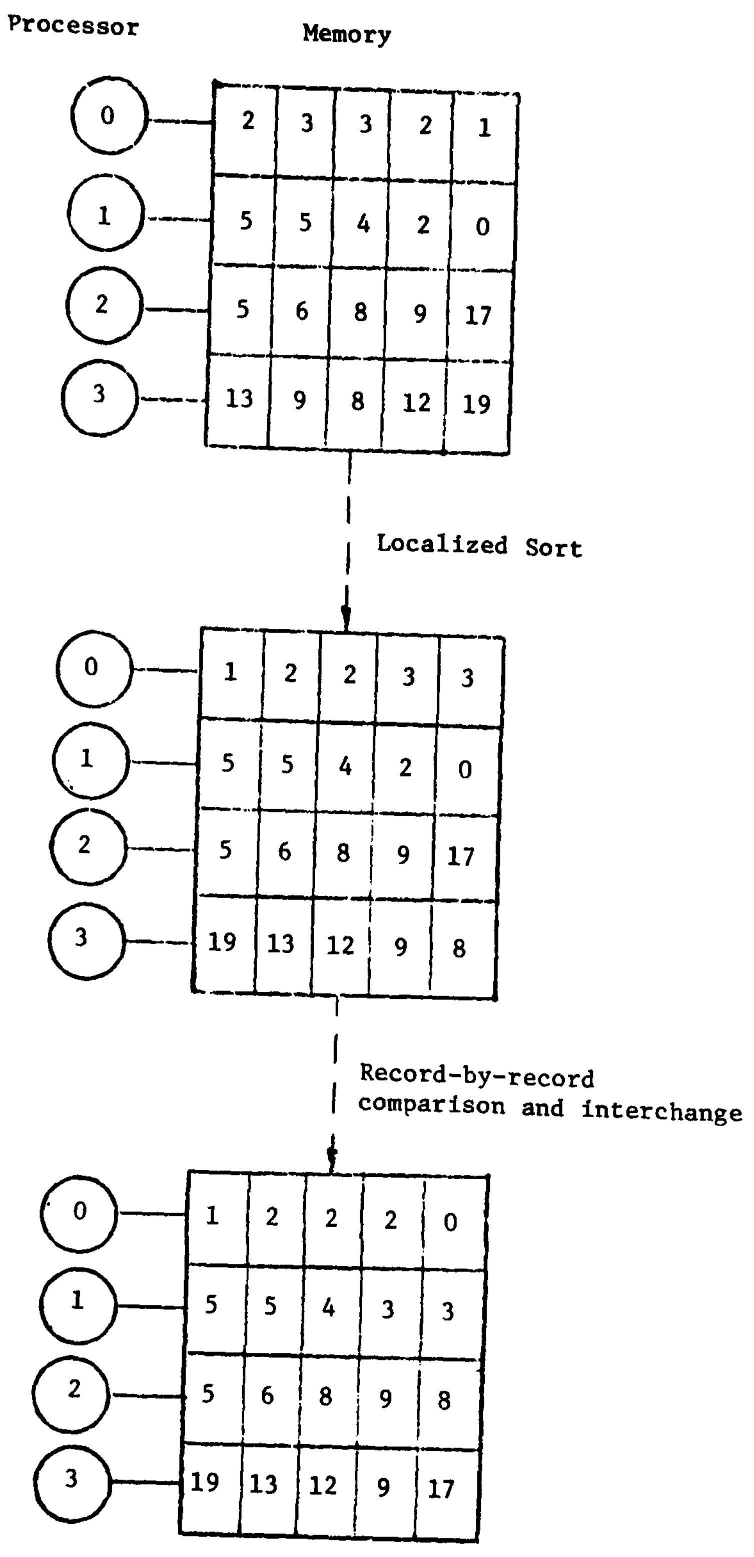


FIGURE 7c. Step 3: EXCHANGE [0,1] and EXCHANGE [2,3]

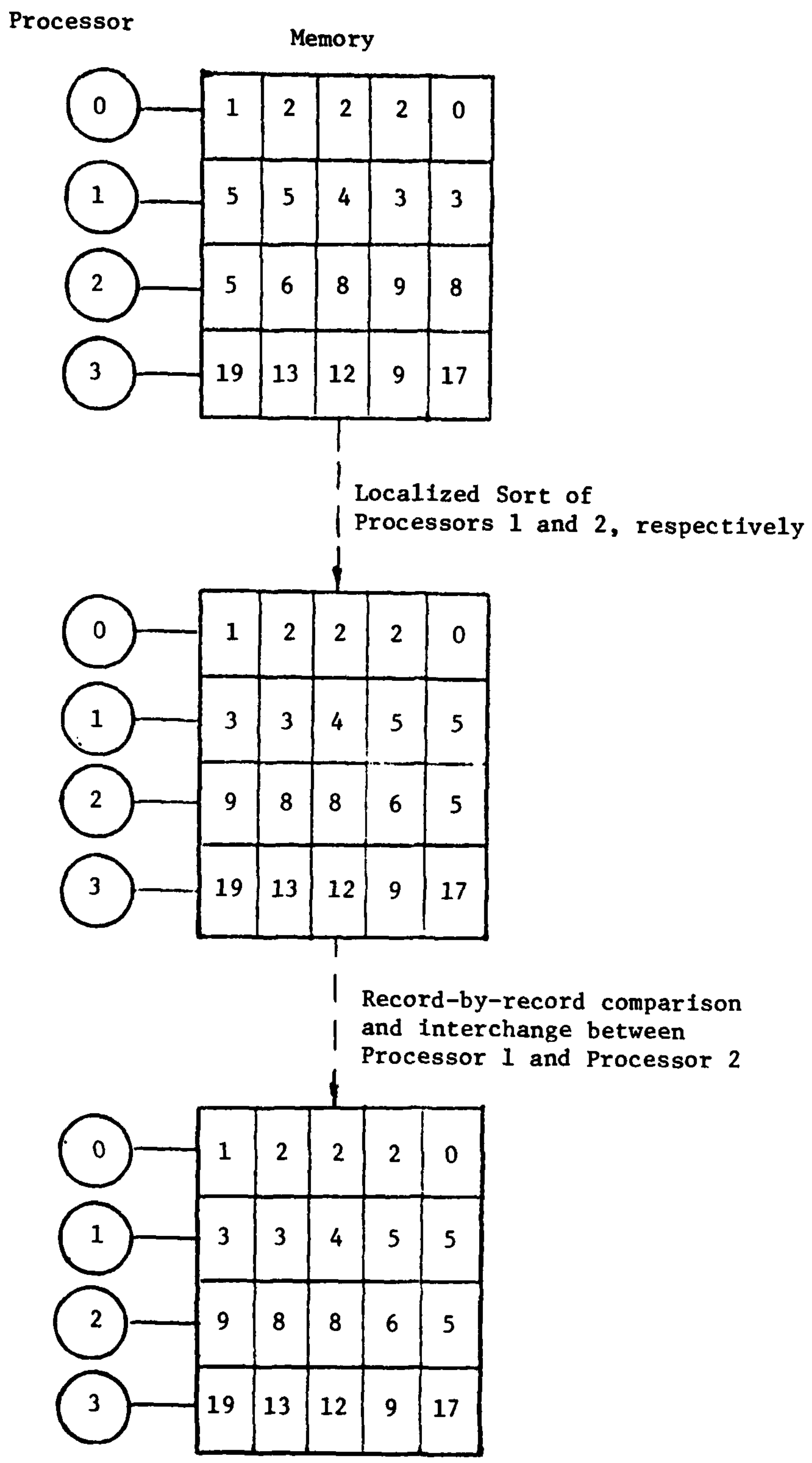


FIGURE 7d. Step 4: EXCHANGE [1,2]

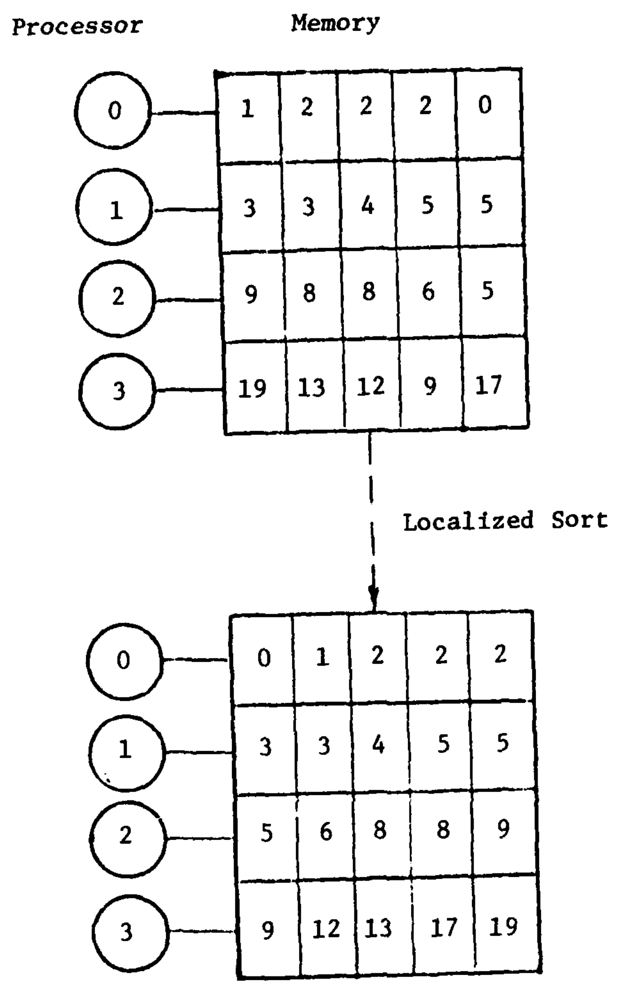


FIGURE 7e. Final step: Localized Sort in non-decreasing order

stitute the parallel sorting algorithm. P is the number of processors, M is the number of records per processor and Step is a variable for the step number.

A. The Procedure Executed by the Post Processing Controller

```
Step = 1
while Step ≤ P
do
   Broadcast ('Exchange', Step)
   Step = Step + 1
end
Broadcast ('localized sort')
```

In the algorithm, there are P exchanges and one localized sort. When the PPC broadcasts an Exchange command to the P processors, it includes, as argument, the step number.

B. Procedures Executed by a Processor

Processor i executes an 'Exchange' command as follows.

```
Procedure Exchange (Step) /* as executed by Processor i */
         If Step is even and i = Ø then stop
         If Step is even and P is even and i = P-1 then stop
         If Step is odd and P is odd and i = P-1 then stop
         If Step is odd then
            do if i is even then j = i + 1 else j = i - 1
            end
         If Step is even then
            do if i is even then j = i - 1 else j = i + 1
            end
         If j = i + 1 then
            do sort all M records in non-decreasing order
               Send (j)
            end
         else
            do sort all M records in non-increasing order
               Receive (j)
            end
end Procedure Exchange
```

The procedures Send and Receive are defined as follows.

```
Procedure Send (j) /* as executed by Processor i */
   Count = 1 /* count of records in Processor i or j */
   while Count ≤ M
```

```
    do send the next record R[i, Count] to Processor j
       wait for the return record from j
       call this record R[i, Count]
       Count = Count + 1
    end
end Procedure Send

Procedure Receive (j) /* as executed by Processor i */
    Count = 1
    while Count ≤ M
    do wait for the next record R[j, Count] from Processor j
       compare this record with own next record R[i, Count]
    If the sort value of R[i, Count] is smaller, then
       interchange R[i, Count] with R[j, Count]
    send R[j, Count] back to Processor j
    Count = Count + 1
    end
end Procedure Receive
```

Whenever Processors i and j need to exchange records, then the one which ought to keep the smaller records executes its Send procedure after sorting its own records in non-decreasing order. The other processor executes its Receive procedure after sorting its own records in non-increasing order. If Processor i ought to keep the smaller records, then it sends its records, one at a time, to Processor j. Processor j then keeps the bigger records (after record by record comparison) and sends the smaller ones back to Processor i.

```
Procedure Localized Sort
    Sort all M records in non-decreasing order
end Procedure Localized Sort
```

This is the procedure executed by each processor in response to the 'localized sort' command from the PPC. Localized sorting is also needed during the execution of the Exchange procedure. Localized sorting in the first step of the algorithm may be accomplished by using the merge-sort method in [11, pages 159-168]. This method requires 2M memory to sort M records. Localized sorting during any other step involves sorting a bitonic sequence, and this can be done by simply merging records starting at the two ends.

5.3 Interconnection of Processors

It is easy to see that, in this algorithm, each Processor i, for $1 \le i \le P-2$, interacts directly only with Processors $i+1$ and $i-1$. Processor 0 only interacts with Processor 1, and Processor P-1 interacts only with Processor P-2. Therefore, for each processor, we need only connect directly to a maximum of two other processors, the one 'in front of' it, and the one 'behind' it. Figure 8 shows the nature of interconnections for various numbers of processors. We note, at this point, that there is no restriction on the number of processors, i.e., the value of P. P can be any positive number. Also, since each processor needs to be connected directly only to two others, we have a hardware structure which is easily extensible.

5.4 Analysis of Time Complexity

Let r represent the amount of time required to route a single record from one processor to another. Let c denote the time taken by a processor to compare (and interchange, if necessary) two records. There are P processors and M records.

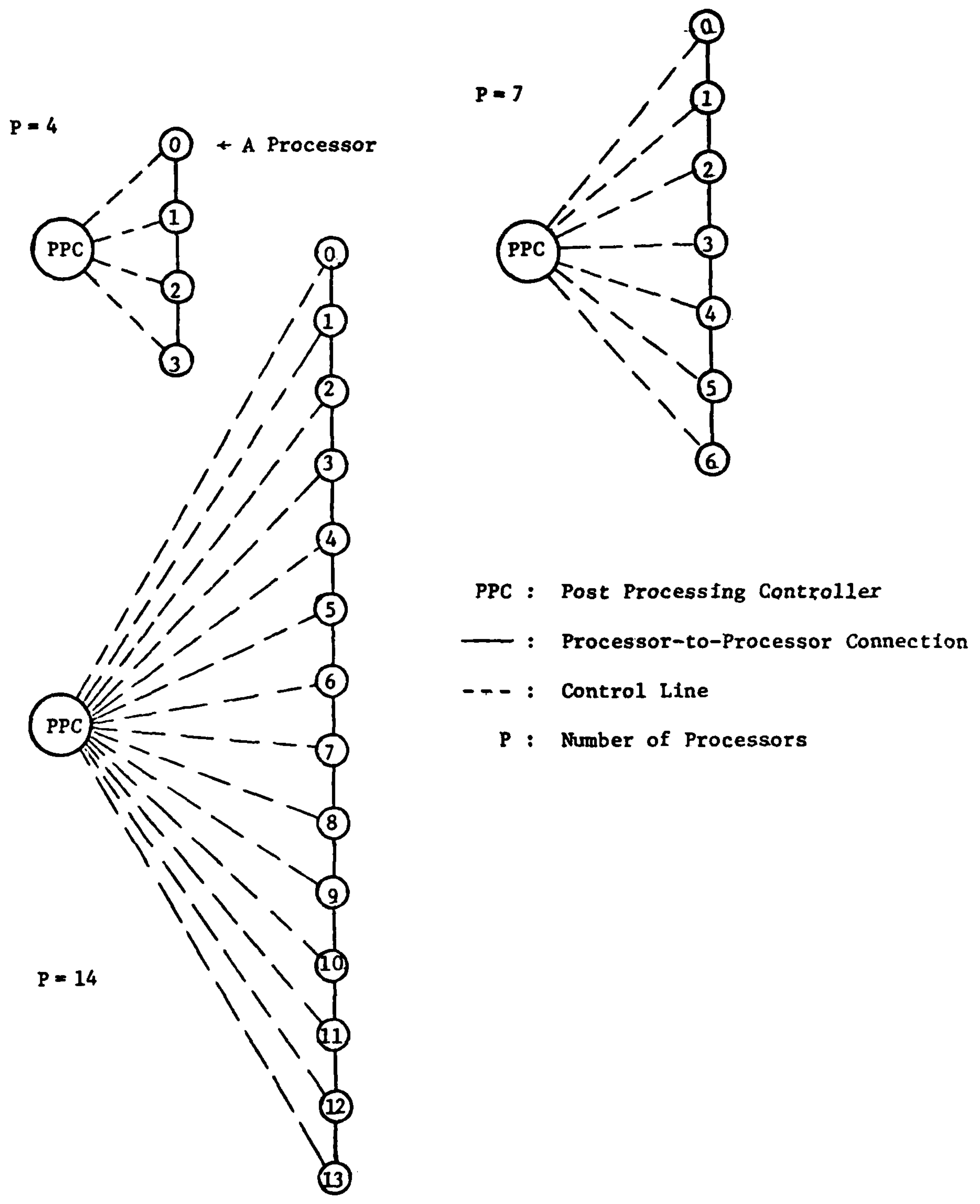
The algorithm consists of P exchange steps followed by one localized sort. The amount of time required by a processor to do localized sorting of a non-bitonic sequence is $(M \log M)c$. The amount of time required by a processor to do localized sorting of a bitonic sequence is Mc, since sorting can be done by simply merging the records from the two ends.

An EXCHANGE is a two-step process. The first step is a localized sort. The second step consists of 2M record-routing operations and M comparisons (and interchanges, if necessary). Therefore, the second step takes $2Mr + Mc$ time units. The time taken for the first exchange step is $(M \log M)c + 2\,Mr + Mc$ time units since it involves sorting a non-bitonic sequence. The time taken for each of the other $(P-1)$ exchange steps is $2Mr + 2Mc$ time units. The time taken for the final localized sort is Mc time units. Hence, the total time taken by the algorithm is

$$\begin{aligned}&(M \log M)c + 2Mr + Mc + (P-1)(2Mc + 2Mr) + Mc\\ &\quad = c(M \log M) + (2c + 2r)\,(MP)\end{aligned}$$

Hence, the order of time is given by

$$O(M \log M + MP)$$



PPC :	Post Processing Controller
——— :	Processor-to-Processor Connection
- - - :	Control Line
P :	Number of Processors

FIGURE 8. Interconnection of Processors for odd-even sort.

In our design, we expect that $M >> P$, so that $\log M >> P$. Therefore, the sorting time approaches $O(M \log M)$, and this is the best that can be expected from P processors.

5.5 Analysis of Processor Utilization

We shall use the concept of efficiency as defined in [8]. The _efficiency of a processor_ in the execution of a given task is defined as the ratio between the share of the work H actually performed by the processor in unit time and the work W it would have performed in unit time if it were acting alone.

In the context of sorting algorithms, let us say that a single processor, acting alone, can sort MP elements in $cMP (\log (MP))$ units of time, where c has already been defined before. That is, the processor performs $1/(cMP \log (MP))$ units of work in unit time. That is,

$$W = 1/(cMP \log (MP)).$$

Our sorting algorithm sorts MP elements using P parallel processors in a time given by

$$cM \log M + dMP$$

where $d = (2c + 2r)$. Thus, the amount of work performed by any one of the P processors in unit time is given by

$$H = (1/P)(1/(cM \log M + dMP))$$

Hence, the efficiency E of a processor in this algorithm is

$$E = H/W = (cM \log (MP))/(cM \log M + dMP).$$

For clarity, this may be rewritten as

$$E = (cM \log M + cM \log P)/(cM \log M + dMP).$$

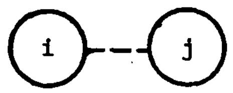
From the above equation, it is clear that the efficiency becomes very close to one for large values of M.

6. METHOD II - THE MODIFIED STONE SORT

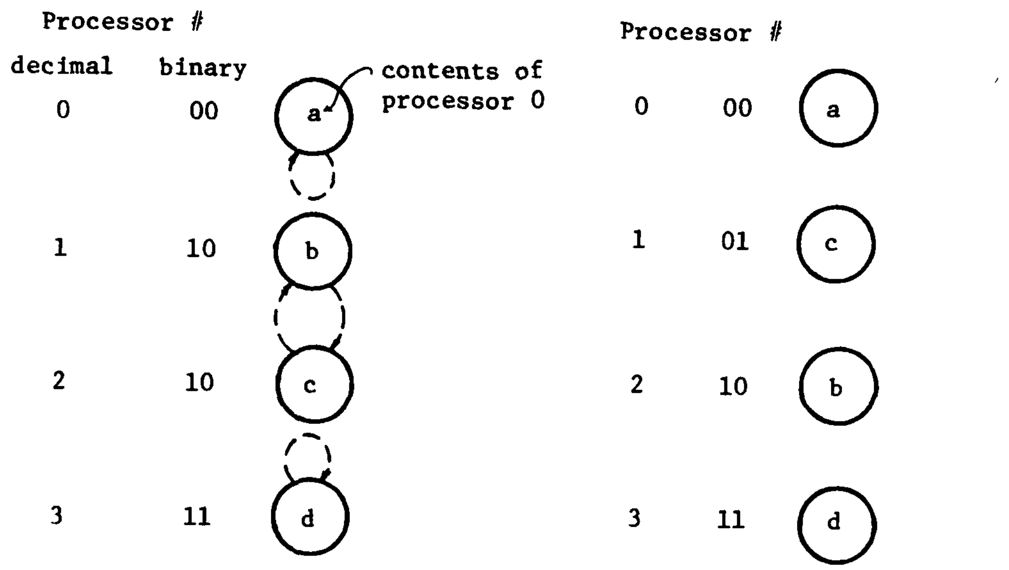
This is based upon the method used by Stone [15] to sort P elements in $\log^2 P$ time using P processors. His method sorts P elements, where P is a power of two, using P processors in the following manner. The sorting proceeds in log P stages of log P steps each. Each step consists of a perfect shuffle followed by an operation performed on P/2 pairs of adjacent elements. We will describe the perfect shuffle and the operation that follows it, in turn.

Remember that the number of processors is P, and that P is a power of two. The processors are numbered 0, 1, 2, ..., (P-1). To represent the processors in binary notation, it requires log P bits. Thus, if $P = 4$, the processors may be represented as Processor 00, Processor 01, Processor 10, and Processor 11 in binary notation, or as Processor 0, Processor 1, Processor 2 and Processor 3 in decimal notation. Let ibin be the binary representation of a decimal number i. Also, let ibin´ be the result of left circularly shifting by one bit the binary number ibin. Thus, if ibin = 001, then ibin´ = 010. On the other hand, if ibin = 111, then ibin´ = 111. It is also true that ibin = ibin´ = 000. Now, we say that P processors perform a _perfect shuffle_ if, Processor i, for $0 \le i \le (P-1)$, moves the contents of its memory to the memory of Processor j such that ibin´ = jbin. We shall say that Processor j is the _shuffle processor_ of Processor i. Also, Processor i is the _reverse shuffle processor_ of Processor j. Therefore, in a perfect shuffle, each processor moves its contents into its shuffle processor, and receives the contents of its reverse shuffle processor. A perfect shuffle operation involving four processors is shown in Figure 9.

Each perfect shuffle is followed by an operation performed between pairs of adjacent processors, say, i and j. The operation may be one of the following three: (1) no operation. (2) A comparison of the element stored in Processor i's memory with the element stored in Processor j's memory and the storing of the smaller of the two elements in Processor i's memory and the larger of the two elements in Processor j's memory. This operation will correspond to an EXCHANGE [i,j] in our modified parallel sort, or (3) a comparison of the element stored in Processor i's memory with the element stored in Processor j's memory, and the storing of the larger of the two elements in Processor i's memory and the smaller of the



Indicates that j is the shuffle processor of i



Contents of processors before the perfect shuffle

Contents of processors after the perfect shuffle

Processor #	Shuffle Processor #	Reverse Shuffle Processor #
0	0	0
1	2	2
2	1	1
3	3	3

FIGURE 9. An example of a perfect shuffle operation involving four processors.

two elements in Processor j's memory. In our terminology, operation (2) is written as EXCHANGE [i,j] and operation (3) as EXCHANGE [j,i]. The final step in our modified algorithm is a localized sort performed by all the processors.

6.1 An Illustration

The manner in which the P parallel processors sort MP records is illustrated by means of an example developed in Figure 10. The initial configuration of records and the configuration at the end of each step are shown in these figures. As can be seen, $P = 4$ and $M = 5$. The process consists of two stages, where each stage consists of two steps. A final localized sorting step is also involved.

STAGE 1

Step 1: Perform the perfect shuffle, i.e., each processor sends the records in its memory to the memory of its shuffle processor. (See Figure 10a).

Step 2: Perform the perfect shuffle.
EXCHANGE [0, 1], EXCHANGE [3, 2]. (See Figure 10b).

STAGE 2

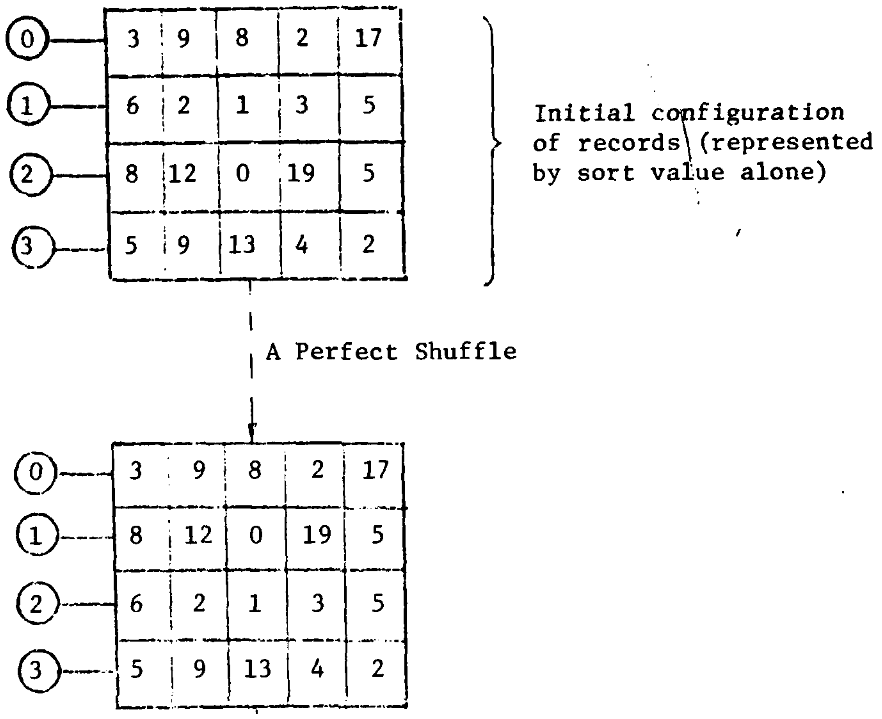
Step 1: Perform the perfect shuffle.
EXCHANGE [0, 1], EXCHANGE [2, 3]. (See Figure 10c).

Step 2: Perform the perfect shuffle.
EXCHANGE [0, 1], EXCHANGE [2, 3]. (See Figure 10d).

Final Step: Perform localized sort in all processors in non-decreasing order. (See Figure 10e).

6.2 The Algorithm

Once Again, the algorithm is coordinated by the post processing controller (PPC) which broadcasts commands to all processors. In the procedures, P is the number of processors, M is the number of records per processor, Step is a variable for the step number, and Stage is a variable for the stage number.



0
1
2
3
Initial configuration of records (represented by sort value alone)
A Perfect Shuffle
0
1
2
3

0	3	9	8	2	17
1	6	2	1	3	5
2	8	12	0	19	5
3	5	9	13	4	2

0	3	9	8	2	17
1	8	12	0	19	5
2	6	2	1	3	5
3	5	9	13	4	2

FIGURE 10a: An Illustration of the Modified Stone Sort - Step 1 of Stage 1

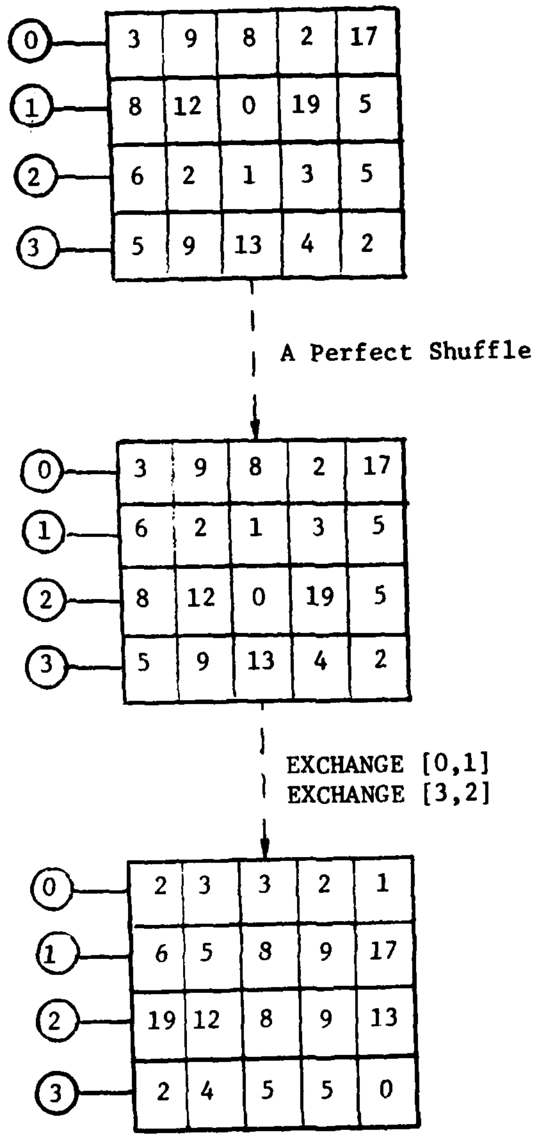


FIGURE 10b: An Illustration of the Modified Stone Sort - Step 2 of Stage 1

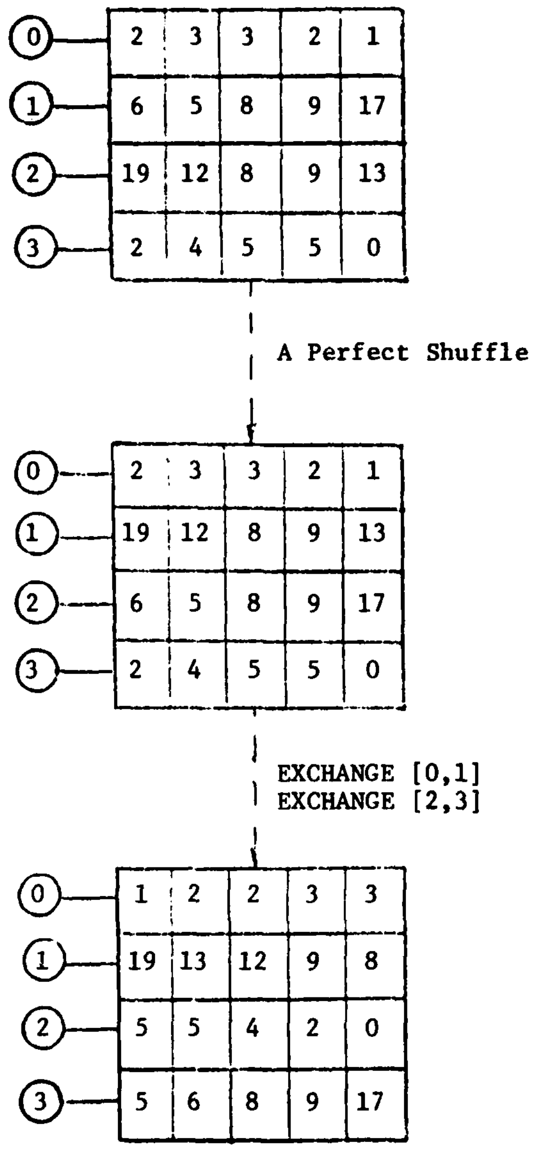


FIGURE 10c: An Illustration of the Modified Stone Sort - Step 1 of Stage 2

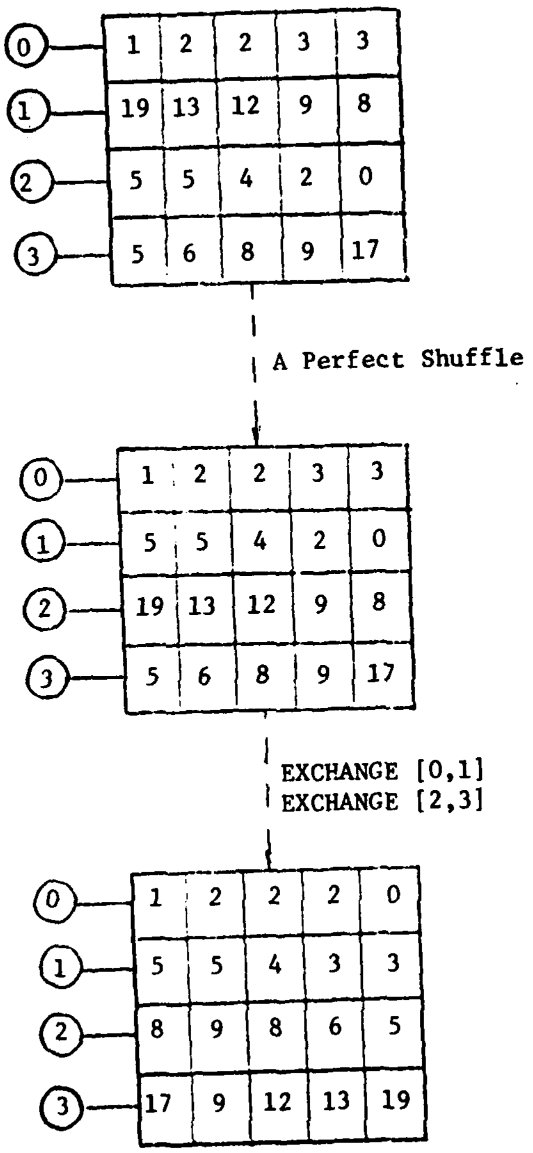


FIGURE 10d: An Illustration of the Modified Stone Sort - Step 2 of Stage 2

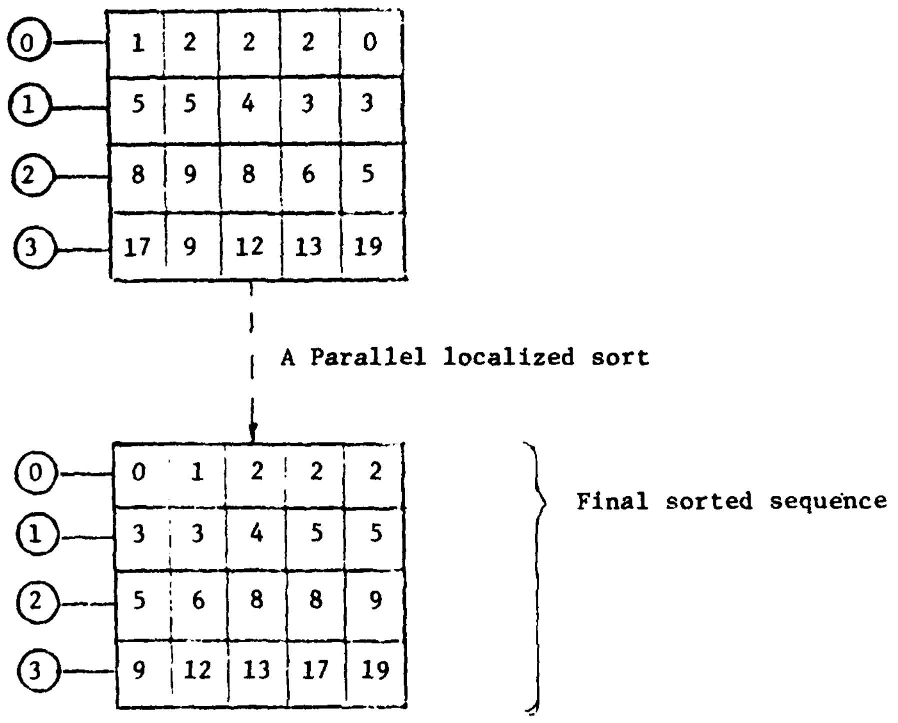


FIGURE 10e: An Illustration of the Modified Stone Sort - Final localized sort step.

A. The Procedure Executed by the Post Processing Controller

```
Stage = 1
while Stage ≤ log P
do Step = 1
   while Step ≤ log P
   do Broadcast ('Shuffle')
      Broadcast ('Exchange', Stage, Step)
      Step = Step + 1
   end
   Stage = Stage + 1
end
Broadcast ('localized sort')
```

The algorithm consists of $\log^2 P$ 'Shuffle' and 'Exchange' commands and one 'localized sort' command. Step and Stage are passed as arguments of the 'Exchange' command.

B. Procedures Executed by a Processor

Processor i executes an 'Exchange' command as follows.

```
Procedure Exchange (Stage, Step)
   If Stage < log P
      Then do
         If Step ≤ (log P - Stage)
            then Stop /* the 'Ø' process */
            else do q = Step - log P + Stage
                    r = i Mod ($2^{q+1}$)
                    if r is even and r < $2^q$
                       then do sort the M records in
                               non-decreasing order.
                               Send (i + 1)
                            end
                       else if r is odd and r > $2^q$
                               then do Sort the M records
                                       in non-decreasing order.
                                       Send (i - 1)
                                    end
                               else if r is even
                                       then do Sort the M records
                                               in non-increasing
                                               order
                                               Receive (i + 1)
                                            end
                                       else do Sort the M records
                                               in non-increasing
                                               order
                                               Receive (i - 1)
                                            end
                 end
         end
```

```
                    else do
                              If i is even
                                 then do Sort the M records in non-decreasing order.
                                         Send (i + 1)
                                      end
                                 else do Sort the M records in non-increasing order.
                                         Receive (i - 1)
                                      end
                          end
          end Procedure Exchange
```

The procedures Send and Receive have been defined earlier.

```
   Procedure Shuffle /* as executed by Processor i */
         ibin = binary equivalent of i
         ibin' = ibin after left circularly shifting by one bit
         ibin'' = ibin after right circularly shifting by one bit
         j = decimal equivalent of ibin'
         k = decimal equivalent of ibin''
         Count = 1 /* Count of records in Processor i or j */
               while Count ≤ M
                  do send the next record R[i, Count] to Processor j
                     wait for next record from Processor k and call it
                        R[i, Count]
                     Count = Count + 1
                  end
    end Procedure Shuffle
```

In procedure Shuffle, each processor places the records that were in its memory into the corresponding location of the shuffle processor's memory and receives the records that were in the memory of the reverse shuffle processor.

```
       Procedure Localized Sort
          Sort all M records in non-decreasing order
       end Procedure Localized Sort
```

This is the procedure executed by each processor in response to the 'localized sort' command from the PPC. Localized sorting in the first stage of the algorithm is done by using a merge-sort method [11, pages 159-168]. Localized sorting during any other step involves sorting a bitonic sequence, and can be done by simply merging records starting at the two ends.

6.3 Interconnection of Processors

From the algorithm, it is easy to see what kind of interconnections are needed between processors. First, to be able to provide the perfect shuffle, each processor must be connected directly to its shuffle processor. We have already indicated how to find the shuffle processor of a particular processor. The procedure is indicated below.

Let ibin = binary notation of decimal i,
ibin′ = ibin after left circular shift by one bit, and
j = decimal equivalent of ibin′. Then
connect Processor i to Processor j.

For P = 8, we have the following interconnections.

Processor 0 is connected to no other processor
Processor 1 is connected to Processor 2
Processor 2 is connected to Processor 4
Processor 3 is connected to Processor 6
Processor 4 is connected to Processor 1
Processor 5 is connected to Processor 3
Processor 6 is connected to Processor 5
Processor 7 is connected to no other processor.

These connections need only be one-way connections. That is, for example, Processor 6 needs to be able to send records to Processor 5, but Processor 5 need not be able to send records to Processor 6.

Also, to provide for the exchange operations, we need that Processor i, $0 \le i \le (P - 1)$, be connected to Processor (i + 1) if i is even, or to Processor (i - 1) if i is odd. These connections, unlike the previous ones, are two-way connections. The algorithm needed to decide on all interconnections (so as to be able to provide for both Shuffles and Exchanges) is shown below.

```
i = 0
while i < P
do ibin = i in binary
   ibin′ = ibin after left circularly shifting by one bit
   j = decimal equivalent of ibin′
   connect Processor i to Processor j
      (if j is different from i)
       by using a one-way connection
```

```
                    if i is even
                       then connect Processor i to Processor (i + 1)
                            using a two-way connection
                       else connect Processor i to Processor (i - 1)
                            using a two-way connection
                       i = i + 1
         end
```

The layout of processors and their interconnections for various values of P are shown in Figure 11.

6.4 Analysis of Time Complexity

Once again, let r represent the amount of time required to route a single record from one processor to another. Let c denote the time required to compare (and interchange, if necessary) two records by the same processor. There are P processors and M records.

The algorithm consists of $\log^2 P$ shuffles. It also contains one exchange step in the first stage, two exchange steps in the second stage, etc., and log P exchange steps in the final stage. Finally, one localized sort is performed by all the processors.

A shuffle operation consists of M record routing operations. Therefore, the entire time spent in shuffling is $rM \log^2 P$.

From the calculations in the previous algorithm (the odd-even sort), we know that the time taken for the first exchange step is

$$c(M \log M) + 2Mr + Mc$$

The time taken for each of the other exchange steps is

$$2Mr + 2Mc$$

Hence, the total time spent in exchanging is

$$(M \log M)c + 2Mr + Mc + (2Mr + 2Mc)(2 + 3 + \dots + \log P)$$
$$= Mc(\log M - 1) + (Mr + Mc)(\log^2 P + \log P)$$

The time taken for the final localized sort is Mc. Therefore, the total time for sorting is

$$Mc(\log M) + (2Mr + Mc) \log^2 P + (Mr + Mc) \log P$$

Hence, the order of time is given by

$$O(M \log M + M \log^2 P)$$

In our design, we expect that $M >> P$, so that $\log M >> \log P$. Therefore, the sorting time approaches $O(M \log M)$.

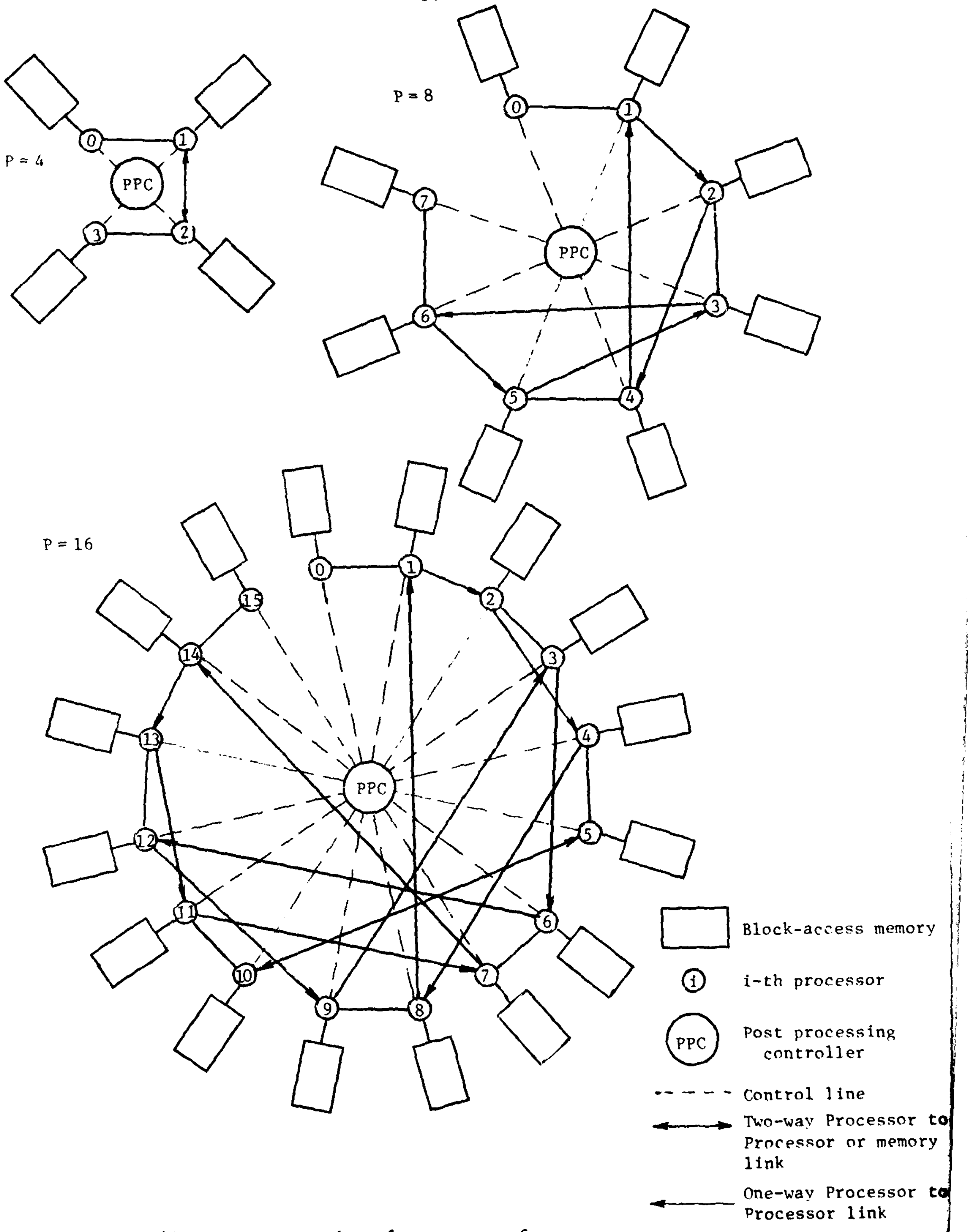


FIGURE 11: Interconnection of processors for sorting using Modified Stone sort

6.5 Analysis of Processor Utilization

Once again, we use the fact that the efficiency of a processor is the ratio between the share H of the work actually performed by the processor in unit time and the work W it would have performed in unit time if it were acting alone.

A single processor can sort MP records in cMP log (MP) units of time. That is,

$$W = 1/cMP \log (MP)$$

Our sorting algorithm sorts MP elements using P parallel processors in a time given by

$$c(M \log M) + dM \log^2 P$$

where d is a constant. That is,

$$H = (1/P)\ (1/(cM \log M + dM \log^2 P))$$

Hence, the efficiency E of a processor in this algorithm is

$$\begin{aligned} E &= H/W \\ &= (c \log (MP))/(c \log M + d \log^2 P) \\ &= (c \log M + c \log P)/(c \log M + d \log^2 P) \end{aligned}$$

Once again, we observe that the efficiency approaches unity as M becomes larger and larger.

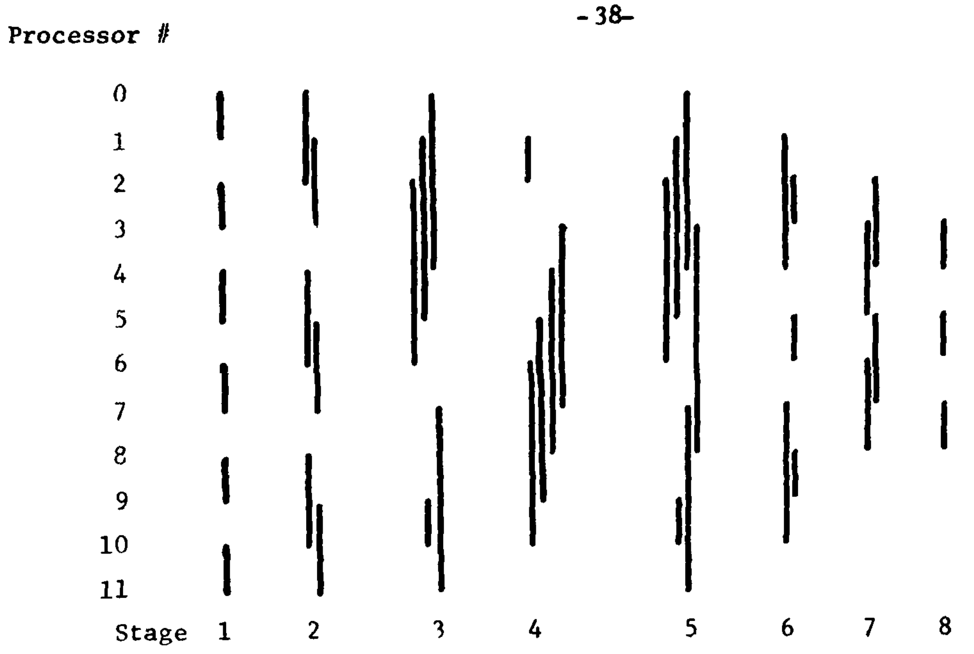
7. METHOD III - THE SPECIALIZED MINIMUM-TIME SORT

So far we have considered two different parallel sorting methods. The odd-even sort is general in that it is applicable for all values of P (the number of processors in the sorter). The modified Stone sort is not so general in that it is applicable only for values of P which are powers of two. Similarly, the method suggested in [8] was also not so general, since it is also applicable only for values of P which are powers of two. In this section, we shall consider a specialized parallel sorting method which is applicable only for specific values of P.

In Figure 12, we have shown two specialized sorting schemes, one of which is applicable only if P = 12, and the other of which is applicable only if P = 16. These schemes cannot be generalized to hold for all values of P.

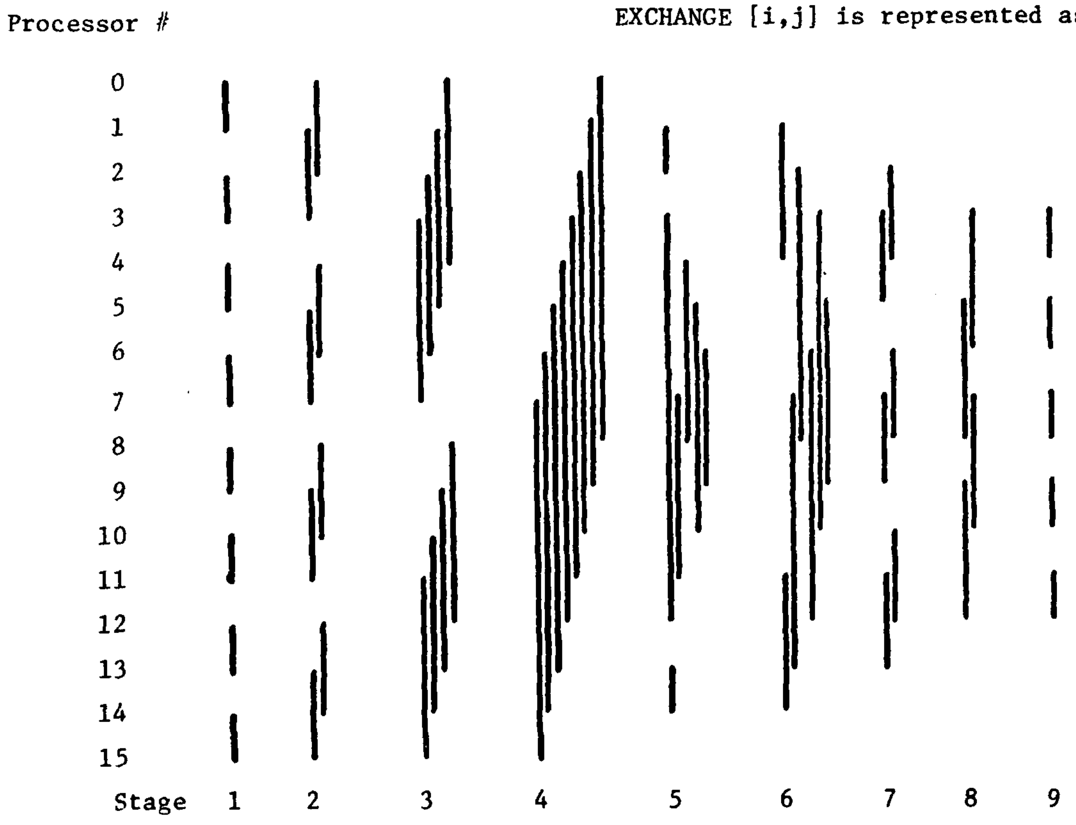
7.1 Motivation for Specialization

The reader might then wonder why we wish to talk of such specialized sorting schemes. Our motivation is as follows. The two methods



P = 12

EXCHANGE [i,j] is represented as $\Big|^{i}_{j}$



P = 16

FIGURE 12: Specialized minimum-time sorts

discussed in previous sections and the method discussed in [8] are not optimal for all values of P. For example, in the case where P = 16, the sorting scheme described by the network in Figure 12 is the optimal one [11, pages 230-231] and will perform better than either of the two methods suggested in this report or the method suggested in [8]. Therefore, if we are going to design our parallel sorter with 16 parallel processors, <u>and we have no intention of increasing the number of processors at a later time</u>, then we would wish to implement the algorithm described by the network in Figure 12.

7.2 Analysis of Time Complexity and Comparison with Others

The network shown in Figure 12, for the case of P = 16, consists of nine stages of exchanges followed by a final localized sort which is not shown in the figure. For example, the first stage consists of EXCHANGE [0, 1], EXCHANGE [2, 3], EXCHANGE [4, 5], EXCHANGE [6, 7], EXCHANGE [8, 9], EXCHANGE [10, 11], EXCHANGE [12, 13], and EXCHANGE [14, 15]. We have already shown that the first stage of exchanges takes $(M \log M)c + 2Mr + Mc$ time units. The remaining eight stages each take $2Mr + 2Mc$ time units. The final localized sort takes Mc time units. Therefore, the total time taken for sorting is

$$cM \log M + 18Mr + 18Mc$$

time units. Compare this with the

$$cM \log M + 32Mr + 32Mc$$

time units taken by the odd-even sort, or the

$$cM \log M + 36Mr + 20Mc$$

time units taken by the modified Stone sort. Thus, the network in Figure 12 is optimal for P = 16.

7.3 Processor Utilization and Limitation

Given a value of P, it will be often possible to develop a specialized algorithm which gives better performance than any of the three generalized methods suggested so far. The trouble with this specialized scheme is, of course, that the value of P cannot be increased at a later time without redesigning the sorting algorithm.

8. CONCLUSIONS

Three sorting methods have been described in this report. The first one, called the odd-even sort, uses P processors, each of which utilizes block-access memory to accommodate M records and two interconnections, to sort MP records in $O(M \log M + MP)$ time. There is no restriction on either P or M. More importantly, since each processor needs to be connected to only two other processors, we can increase the number of processors in the sorter without having to increase the number of connections.

The second method, called the modified Stone sort, uses P processors with block-access memory to sort MP records in $O(M \log M + M \log^2 P)$ time. However, P must be a power of two. This method also requires that each processor be connected only to a maximum of two others, one connection being a one-way link and the other connection being a two-way link.

Finally, we discussed a method, known as the specialized minimum-time sort. This method also uses P processors to sort MP records and completes the sorting in $O(M \log M)$ time, but it can only be used for specific values of P. That is, the algorithms of the method are specialized, and individual algorithms will not work for different values of P. Although algorithms are optimal for given values of P, the architecture of these types of hardware sorters cannot be expanded, since the number of processors is fixed for the original design and optimization.

In all the above methods, the number of records that can be sorted in a batch is restricted only by the memory size of each processor and not by the number of processors.

REFERENCES

[1] Banerjee, J. and Hsiao, D. K., "Performance Evaluation of a Database Computer in Supporting Relational Databases," Proceedings of the Fourth International Conference on Very Large Data Bases, Berlin, Federal Republic of Germany, September 1978, pp. 319-329; and Banerjee, J. and Hsiao, D. K., "The Use of a Database Machine for Supporting Relational Databases," Fourth Workshop on Computer Architecture for Non-numeric Processing, Syracuse, New York, August 1978, pp. 91-98; Also available in Banerjee, J. and Hsiao, D. K., "DBC Software Requirements for Supporting Relational Databases," Technical Report OSU-CISRC-TR-77-7, The Ohio State University, Columbus, Ohio, November 1977.

[2] Banerjee, J., and Hsiao, D. K., "A Methodology for Supporting Existing CODASYL Databases with New Database Machines," Proceedings of the ACM '78 Conference, December 1978; Also available in Banerjee, J., Hsiao, D. K., and Kerr, D. S., "DBC Software Requirements for Supporting Network Databases," Technical Report OSU-CISRC-TR-77-4, The Ohio State University, Columbus, Ohio, June 1977.

[3] Banerjee, J., Hsiao, D. K., and Ng, F. K., "Data Network - A Computer Network of General-Purpose Front-end Computers and Special-Purpose Back-end Database Machines," Proceedings of the International Symposium on Computer Network Protocols, (Danthine, A., Editor), Liege, Belgium, February 1978, pp. D6-1 to D6-12; Also available in Hsiao, D. K., Kerr, D. S., and Ng, F. K., "DBC Software Requirements for Supporting Hierarchical Databases," Technical Report OSU-CISRC-TR-77-1, The Ohio State University, Columbus, Ohio, April 1977.

[4] Banerjee, J. and Hsiao, D. K., "Concepts and Capabilities of a Database Computer," ACM Transactions on Database Systems, Vol. 3, No. 4, December 1978, pp. 347-384. Also available in Baum, R. I., Hsiao, D. K. and Kannan, K., "The Architecture of a Database Computer -- Part I: Concepts and Capabilities," Technical Report OSU-CISRC-TR-76-1, The Ohio State University, Columbus, Ohio, September 1978.

[5] Kannan, K., Hsiao, D. K., and Kerr, D. S., "A Microprogrammed Keyword Transformation Unit for a Database Computer," Proceedings of the Tenth Annual Workshop on Microprogramming, October 1977, Niagara Falls, New York, pp. 71-79; and Hsiao, D. K., Kannan, K., and Kerr, D. S., "Structure Memory Designs for a Database Computer," Proceedings of ACM '77 Conference, October 1977, Seattle, Washington, pp. 343-350; Also available in Hsiao, D. K. and Kannan, K., "The Architecture of a Database Computer -- Part II: The Design of the Structure Memory and its Related Processors," Technical Report OSU-CISRC-TR-76-2, The Ohio State University, Columbus, Ohio, October 1976.

[6] Banerjee, J., Hsiao, D. K., and Ng, F., "Database Transformation, Query Translation and Performance Analysis of a New Database Computer in Supporting Hierarchical Database Management," IEEE Transactions on Software Engineering, SE-6,1, January 1980, pp. 91-109.

[7] Kannan, K., "The Design of a Mass Memory for a Database Computer," Proceedings of the Fifth Annual Symposium on Computer Architecture, April 1978, Palo Alto, California, pp. 44-50; Also available in Hsiao, D. K. and Kannan, K., "The Architecture of a Database Computer -- Part III: The Design of the Mass Memory and its Related Processors," Technical Report OSU-CISRC-TR-76-3, The Ohio State University, Columbus, Ohio, December 1976.

[8] Banerjee, J. and Hsiao, D. K., "Parallel Bitonic Record Sort - An Effective Algorithm for the Realization of a Post Processor," Technical Report OSU-CISRC-TR-79-1, The Ohio State University, Columbus, Ohio, April 1979.

[9] Menon, M. J. and Hsiao, D. K., "The Access Control Mechanism of a Database Computer (DBC)," Fifth Workshop on Computer Architecture for Non-numeric Processing, Asilomar, California, March 1980; Also available in Banerjee, J., Hsiao, D. K., and Menon, M. J., "The Clustering and Security Mechanisms of a Database Computer (DBC)," Technical Report OSU-CISRC-TR-79-2, The Ohio State University, Columbus, Ohio, April 1979.

[10] Hsiao, D. K. and Menon, M. J., "Design and Analysis of Update Mechanisms of a Database Computer (DBC)," Technical Report OSU-CISRC-TR-80-3, The Ohio State University, Columbus, Ohio, June 1980.

[11] Knuth, D. E., Sorting and Searching, The Art of Computer Programming, Vol. 3, Addison-Wesley, Reading, Massachusetts, 1973.

[12] Chen, T. C., Lum, V. Y., and Tung, C., "The Rebound Sorter: An Efficient Sort Engine for Large Files," Proceedings of the Fourth International Conference on Very Large Data Bases, West Berlin, Germany, September 1978, pp. 312-318.

[13] Nassimi, D. and Sahni, S., "Bitonic Sort on a Mesh-Connected Parallel Computer," IEEE Transactions on Computers, Vol. C-27, No. 1, January 1979, pp. 2-7.

[14] Thompson, C. D. and Kung, H. T., "Sorting on a Mesh-Connected Parallel Computer," Communications of the ACM, Vol. 20, No. 4, April 1977, pp. 263-271.

[15] Stone, H. S., "Parallel Processing with the Perfect Shuffle," IEEE Transactions on Computers, Vol. C-20, 1971, pp. 153-161.

[16] Edelberg, M. and Schissler, L. R., "Intelligent Memory," AFIPS Conference Proceedings, Vol. 45, 1976, pp. 393-400.

[17] Baudet, G. and Stevenson, D., "Optimal Sorting Algorithms for Parallel Computers," IEEE Transactions on Computers, Vol. C-27, No. 1, January 1978, pp. 84-87.

[18] Alekseyev, V. E., Kibernetica, Vol. 5, No. 5, 1969, pp. 99-103.